Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

**Курсовой проект по курсу**

**«Операционные системы»**

Студент: Останина Анна Андреевна

Группа: М8О-208Б-22

Вариант: 16

Преподаватель: Миронов Евгений Сергеевич

Оценка: \_\_\_\_\_\_\_\_\_\_\_

Дата: \_\_\_\_\_\_\_\_\_\_\_

Подпись: \_\_\_\_\_\_\_\_\_\_\_

Москва, 2024

**Содержание**

1. Репозиторий
2. Цель работы
3. Задание
4. Описание работы программы
5. Исходный код
6. Консоль
7. Графики
8. Выводы

### Репозиторий

<https://github.com/Imariiii/os_labs>

### Цель работы

* Приобретение практических навыков в использовании знаний, полученных в течении курса.
* Проведение исследования в выбранной предметной области.

### Задание

Необходимо сравнить два алгоритма аллокации: выделение памяти по степени 2 и алгоритм Мак-Кьюзи-Кэрелса

### Описание работы программы

Каждый аллокатор должен обладать следующим интерфейсом (могут быть отличия в зависимости от особенностей алгоритма):

* Allocator createMemoryAllocator (void realMemory, size\_t memory\_size) - создание аллокатора памяти размера memory\_size.
* void\* alloc(Allocator \* allocator, size\_t block\_size) - выделение памяти при помощи аллокатора размера block\_size.
* void\* free(Allocator \* allocator, void \* block) - возвращает выделенную память аллокатору.

***Выделение памяти по степени 2:***

* Вся память делится на списки свободных элементов равного размера.
* Если элемент свободен, то он хранит ссылку на следующий свободный элемент.
* Если элемент занят, то хранится ссылка на голову списка, откуда он был взят.
* При освобождении нужно просто добавить элемент к голове указанного списка.
* При выделении выбирается список K = [log2(N)].

***Алгоритм Мак-Кьюзи-Кэрелса:***

Алгоритм подразумевает, что память разбита на набор последовательных страниц, и все буферы, относящиеся к одной странице, должны иметь одинаковый размер (являющийся некоторой степенью числа 2). Каждая страница может находиться в одном из трёх перечисленных состояний.

* Быть свободной.
* Быть разбитой на буферы определённого размера.
* Являться частью буфера, объединяющего сразу несколько страниц.

**Исходный код**

mck\_allocator.h

#ifndef POW\_ALLOCATOR\_H

#define MCK\_ALLOCATOR\_H

#include <stddef.h>

#include <stdbool.h>

**struct** Page {

**struct** Page\* next;

**bool** isLarge;

size\_t blockSize;

};

**struct** MCKAllocator {

**struct** Page\* freePagesList;

size\_t pageSize;

};

**void** MCKInit(**struct** MCKAllocator\* a);

**void** MCKDestroy(**struct** MCKAllocator\* a);

**void** \*MCKAlloc(**struct** MCKAllocator\* a, size\_t newblockSize);

**void** MCKFree(**struct** MCKAllocator\* a, **void**\* block);

#endif // MCK\_ALLOCATOR\_H

mck\_allocator.c

#include "mck\_allocator.h"

#include <sys/mman.h>

#include <unistd.h>

**void** MCKInit(**struct** MCKAllocator\* a) {

a->freePagesList = NULL;

a->pageSize = getpagesize();

}

**void** MCKDestroy(**struct** MCKAllocator\* a) {

**struct** Page\* curPage = a->freePagesList;

**while** (curPage) {

**struct** Page\* toDelete = curPage;

curPage = curPage->next;

munmap(toDelete, a->pageSize);

toDelete = NULL;

}

a->freePagesList = NULL;

}

**void** \*MCKAlloc(**struct** MCKAllocator\* a, size\_t newblockSize) {

size\_t roundedBlockSize = 1;

**while** (roundedBlockSize < newblockSize) {

roundedBlockSize \*= 2;

}

**struct** Page\* curPage = a->freePagesList;

**while** (curPage) {

**if** (!curPage->isLarge &&

curPage->blockSize == roundedBlockSize) {

**void**\* block = (**void**\*)(curPage);

a->freePagesList = curPage->next;

**return** block;

}

curPage = curPage->next;

}

**struct** Page\* newPage =

(**struct** Page\*)(mmap(NULL, a->pageSize, PROT\_READ | PROT\_WRITE,

MAP\_PRIVATE | MAP\_ANONYMOUS, -1, 0));

**if** (newPage == MAP\_FAILED) {

**return** NULL;

}

newPage->isLarge = **false**;

newPage->blockSize = roundedBlockSize;

newPage->next = NULL;

size\_t numBlocks = a->pageSize / roundedBlockSize;

**for** (size\_t i = 0; i != numBlocks; ++i) {

**struct** Page\* blockPage = (**struct** Page\*)(

(**char**\*)newPage + i \* roundedBlockSize);

blockPage->isLarge = **false**;

blockPage->blockSize = roundedBlockSize;

blockPage->next = a->freePagesList;

a->freePagesList = blockPage;

}

**void**\* block = (**void**\*)(newPage);

a->freePagesList = newPage->next;

**return** block;

}

**void** MCKFree(**struct** MCKAllocator\* a, **void**\* block) {

**if** (block == NULL) **return**;

**struct** Page\* page = (**struct** Page\*)(block);

page->next = a->freePagesList;

a->freePagesList = page;

}

pow\_allocator.h

#ifndef POW\_ALLOCATOR\_H

#define POW\_ALLOCATOR\_H

#include <stddef.h>

#define INIT\_POW 5

#define NUM\_OF\_POWS 13

**void** PowInit();

**void** \*PowAlloc(size\_t size);

**void** PowFree(**void** \*ptr);

#endif // POW\_ALLOCATOR\_H

pow\_allocator.c

#include "pow\_allocator.h"

#include <math.h>

#include <stdio.h>

**struct** Elem {

**void** \*ptr;

**char** data[];

};

**static** **void** \*HEADS[NUM\_OF\_POWS];

**static** **char** POOL[100000000];

**void** PowInit() {

**char** \*data = POOL;

**for** (**int** i = 0; i < NUM\_OF\_POWS; ++i) {

HEADS[i] = data;

**int** pow = INIT\_POW;

size\_t sizeofBlock = (1 << pow) + **sizeof**(**void** \*);

**for** (**int** j = 0; j < 100000; ++j) {

((**struct** Elem \*)data)->ptr = data + sizeofBlock;

data += sizeofBlock;

}

((**struct** Elem \*)data)->ptr = NULL;

pow++;

data += sizeofBlock;

}

}

**void** \*PowAlloc(size\_t size) {

**int** k;

**if** (size < (1 << NUM\_OF\_POWS) + 1) {

k = 0;

} **else** {

k = ceil(log2(size)) - INIT\_POW;

}

**struct** Elem \*temp = HEADS[k];

HEADS[k] = temp->ptr;

temp->ptr=HEADS+k;

**return** temp->data;

}

**void** PowFree(**void** \*ptr) {

**struct** Elem \*temp = (**struct** Elem \*)((**char**\*)ptr-**sizeof**(**void**\*));

**void** \*\*headPtr = temp->ptr;

temp->ptr = \*headPtr;

\*headPtr = temp;

}

main.cpp

#include <unistd.h>

#include <chrono>

#include <cstdlib>

#include <iostream>

#include <vector>

**extern** "C" {

#include "mck\_allocator.h"

#include "pow\_allocator.h"

}

size\_t page\_size = sysconf(\_SC\_PAGESIZE);

**static** **void** benchmark(**struct** MCKAllocator\* MCKAlloctor, std::size\_t size,

std::size\_t n) {

PowInit();

MCKInit(MCKAlloctor);

std::vector<**void**\*> list\_blocks;

std::vector<**void**\*> MKC\_blocks;

std::cout << "Comparing PowAllocator and MCKAllocator" << std::endl;

std::cout << "Block allocation rate of " << n << " chunks of " << size

<< " bytes" << std::endl;

**auto** start\_time = std::chrono::steady\_clock::now();

**for** (size\_t i = 0; i != n; ++i) {

**void**\* block = PowAlloc(size);

list\_blocks.push\_back(block);

}

**auto** end\_time = std::chrono::steady\_clock::now();

std::cout << "Time of alloc PowAllocator: "

<< std::chrono::duration\_cast<std::chrono::milliseconds>(

end\_time - start\_time)

.count()

<< " milliseconds" << std::endl;

start\_time = std::chrono::steady\_clock::now();

**for** (size\_t i = 0; i != n; ++i) {

**void**\* block = MCKAlloc(MCKAlloctor, size);

MKC\_blocks.push\_back(block);

}

end\_time = std::chrono::steady\_clock::now();

std::cout << "Time of alloc MCKAllocator: "

<< std::chrono::duration\_cast<std::chrono::milliseconds>(

end\_time - start\_time)

.count()

<< " milliseconds" << std::endl;

std::cout << "Block free rate" << std::endl;

start\_time = std::chrono::steady\_clock::now();

**for** (size\_t i = 0; i != list\_blocks.size(); ++i) {

PowFree(list\_blocks[i]);

}

end\_time = std::chrono::steady\_clock::now();

std::cout << "Time of free PowAllocator: "

<< std::chrono::duration\_cast<std::chrono::milliseconds>(

end\_time - start\_time)

.count()

<< " milliseconds" << std::endl;

start\_time = std::chrono::steady\_clock::now();

**for** (size\_t i = 0; i != MKC\_blocks.size(); ++i) {

MCKFree(MCKAlloctor, MKC\_blocks[i]);

}

end\_time = std::chrono::steady\_clock::now();

std::cout << "Time of free MCKAllocator: "

<< std::chrono::duration\_cast<std::chrono::milliseconds>(

end\_time - start\_time)

.count()

<< " milliseconds" << std::endl;

}

**int** main() {

MCKAllocator a;

**for** (size\_t i = 0; i < 10; ++i)

benchmark(&a, 100\*(i+1), 70000);

std::cout << "-----------------------";

**for** (size\_t i = 0; i < 5; ++i)

benchmark(&a, 1000, 10000\*(i+1));

}

Консоль

anna@anna-**virtual**-machine:~/labs\_3sem/os\_labs/build/cp$ ./cp

Comparing PowAllocator **and** MCKAllocator

Block allocation rate of 70000 chunks of 100 bytes

Time of alloc PowAllocator: 6 milliseconds

Time of alloc MCKAllocator: 815 milliseconds

Block free rate

Time of free PowAllocator: 0 milliseconds

Time of free MCKAllocator: 5 milliseconds

Comparing PowAllocator **and** MCKAllocator

Block allocation rate of 70000 chunks of 200 bytes

Time of alloc PowAllocator: 5 milliseconds

Time of alloc MCKAllocator: 1343 milliseconds

Block free rate

Time of free PowAllocator: 1 milliseconds

Time of free MCKAllocator: 3 milliseconds

Comparing PowAllocator **and** MCKAllocator

Block allocation rate of 70000 chunks of 300 bytes

Time of alloc PowAllocator: 3 milliseconds

Time of alloc MCKAllocator: 1000 milliseconds

Block free rate

Time of free PowAllocator: 1 milliseconds

Time of free MCKAllocator: 3 milliseconds

Comparing PowAllocator **and** MCKAllocator

Block allocation rate of 70000 chunks of 400 bytes

Time of alloc PowAllocator: 4 milliseconds

Time of alloc MCKAllocator: 955 milliseconds

Block free rate

Time of free PowAllocator: 0 milliseconds

Time of free MCKAllocator: 2 milliseconds

Comparing PowAllocator **and** MCKAllocator

Block allocation rate of 70000 chunks of 500 bytes

Time of alloc PowAllocator: 4 milliseconds

Time of alloc MCKAllocator: 948 milliseconds

Block free rate

Time of free PowAllocator: 1 milliseconds

Time of free MCKAllocator: 4 milliseconds

Comparing PowAllocator **and** MCKAllocator

Block allocation rate of 70000 chunks of 600 bytes

Time of alloc PowAllocator: 7 milliseconds

Time of alloc MCKAllocator: 949 milliseconds

Block free rate

Time of free PowAllocator: 1 milliseconds

Time of free MCKAllocator: 3 milliseconds

Comparing PowAllocator **and** MCKAllocator

Block allocation rate of 70000 chunks of 700 bytes

Time of alloc PowAllocator: 3 milliseconds

Time of alloc MCKAllocator: 1138 milliseconds

Block free rate

Time of free PowAllocator: 0 milliseconds

Time of free MCKAllocator: 3 milliseconds

Comparing PowAllocator **and** MCKAllocator

Block allocation rate of 70000 chunks of 800 bytes

Time of alloc PowAllocator: 13 milliseconds

Time of alloc MCKAllocator: 1390 milliseconds

Block free rate

Time of free PowAllocator: 1 milliseconds

Time of free MCKAllocator: 4 milliseconds

Comparing PowAllocator **and** MCKAllocator

Block allocation rate of 70000 chunks of 900 bytes

Time of alloc PowAllocator: 5 milliseconds

Time of alloc MCKAllocator: 1790 milliseconds

Block free rate

Time of free PowAllocator: 1 milliseconds

Time of free MCKAllocator: 15 milliseconds

Comparing PowAllocator **and** MCKAllocator

Block allocation rate of 70000 chunks of 1000 bytes

Time of alloc PowAllocator: 11 milliseconds

Time of alloc MCKAllocator: 5096 milliseconds

Block free rate

Time of free PowAllocator: 4 milliseconds

Time of free MCKAllocator: 27 milliseconds

-----------------------Comparing PowAllocator **and** MCKAllocator

Block allocation rate of 10000 chunks of 1000 bytes

Time of alloc PowAllocator: 2 milliseconds

Time of alloc MCKAllocator: 575 milliseconds

Block free rate

Time of free PowAllocator: 0 milliseconds

Time of free MCKAllocator: 0 milliseconds

Comparing PowAllocator **and** MCKAllocator

Block allocation rate of 20000 chunks of 1000 bytes

Time of alloc PowAllocator: 0 milliseconds

Time of alloc MCKAllocator: 1373 milliseconds

Block free rate

Time of free PowAllocator: 7 milliseconds

Time of free MCKAllocator: 0 milliseconds

Comparing PowAllocator **and** MCKAllocator

Block allocation rate of 30000 chunks of 1000 bytes

Time of alloc PowAllocator: 2 milliseconds

Time of alloc MCKAllocator: 2263 milliseconds

Block free rate

Time of free PowAllocator: 0 milliseconds

Time of free MCKAllocator: 6 milliseconds

Comparing PowAllocator **and** MCKAllocator

Block allocation rate of 40000 chunks of 1000 bytes

Time of alloc PowAllocator: 20 milliseconds

Time of alloc MCKAllocator: 2359 milliseconds

Block free rate

Time of free PowAllocator: 1 milliseconds

Time of free MCKAllocator: 4 milliseconds

Comparing PowAllocator **and** MCKAllocator

Block allocation rate of 50000 chunks of 1000 bytes

Time of alloc PowAllocator: 2 milliseconds

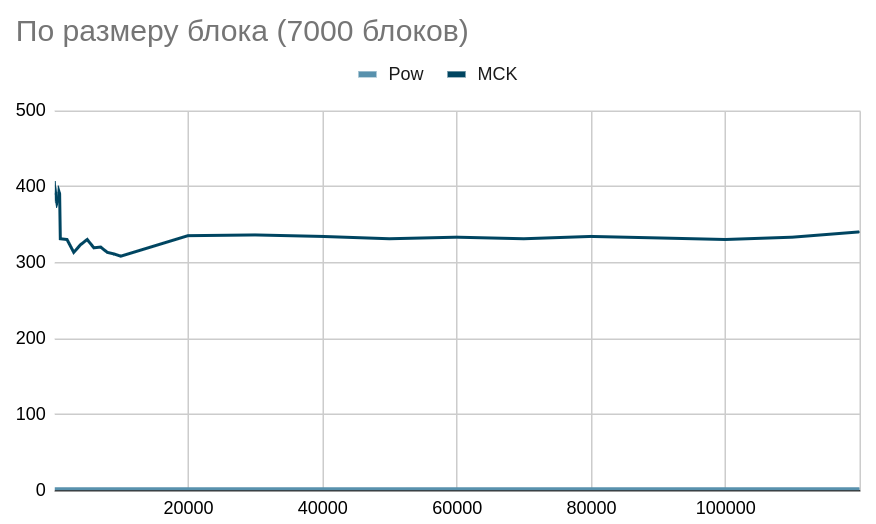
Time of alloc MCKAllocator: 3195 milliseconds

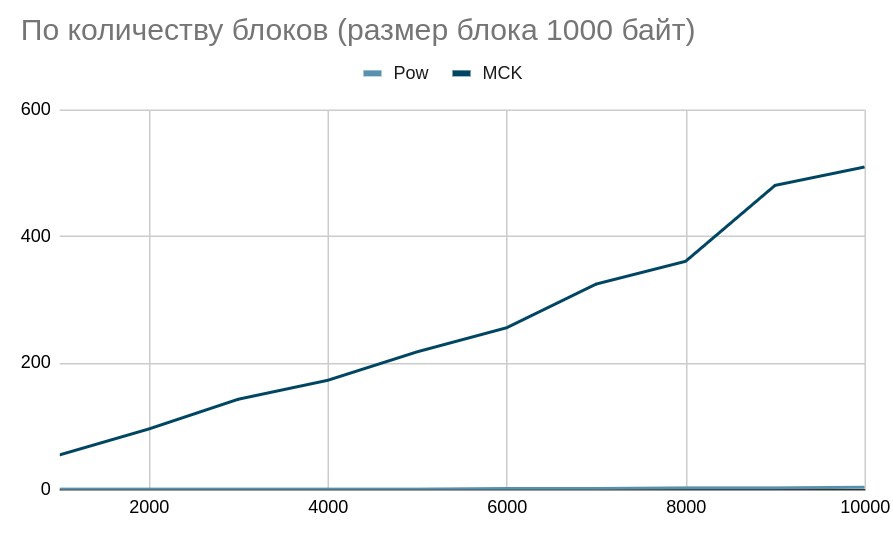
Block free rate

Time of free PowAllocator: 443 milliseconds

Time of free MCKAllocator: 13 milliseconds

### Графики





Выводы

В ходе данного курсового проекта были проведены исследования двух аллокаторов памяти:

Выделение памяти по степени 2 и Алгоритм Мак-Кьюзи-Кэрелса. Сравнивая эти два способа аллокации памяти, основываясь на времени их работы и на самом принципе работы, можно сделать вывод, что метод выделения памяти по степени 2 быстрее, чем алгоритм Мак-Кьюзи-Кэрелса, так как второй алгоритм использует страничный аллокатор, а первый выделяет пул заранее. С другой стороны, второй алгоритм не имеет ограничений по памяти (блоки можно выделять нужного размера, и они формируются по требованиям системы) и выделяет новые страницы только при необходимости. Оба алгоритма не требуют указывать размер освобождаемого блока.